Unidad II: Programación orientada a objetos básica

Lic. Ronaldo Armando Canizales Turcios

Departamento de Electrónica e Informática, UCA

Ciclo virtual 01-2021

Agenda

1. **Elementos de una clase**
   - Métodos
   - Constructores
   - Atributos
   - Setter's y getter's
   - Propiedades
   - Ejemplo
2. **Consejos en el diseño de clases**
   - Ejemplo
3. **UML: Lenguaje unificado de modelado**
   - Ejemplo
4. **Anuncios**

1. Elementos de una clase

Métodos, su definición contiene:

- Un *especificador de acceso* (usualmente public).
- El *tipo de retorno* de dicho método (por ejemplo: void, int, String).
- El nombre del método (usualmente un verbo).
- Una serie de *parámetros* encerrados en paréntesis (opcional).
- *Cuerpo* del método: secuencia de instrucciones encerradas entre llaves.

1. Elementos de una clase

Constructores

- Siempre deben tener el mismo nombre de la clase cuyos objetos está construyendo.
- Se declaran como públicos para permitir poder construir objetos desde cualquier método de nuestra aplicación.
- No tienen un tipo de retorno, pues **no son métodos**.
- El operador **new** invoca al constructor de una clase. No es posible invocar un constructor en un objeto que ya existe

Atributos, su declaración consta de: (1/3)

- Un *especificador de acceso* (usualmente private).
- El *tipo* de variable (por ejemplo: int, String o double).
- El nombre de la variable (usualmente un sustantivo).

Setter's y getter's (2/3)

Es una buena práctica crear métodos llamados **getters** y **setters** para los atributos de nuestras clases. Por ejemplo, si agregáramos para marca y modelo quedarían de la siguiente manera:

```
private String marca;

public String getMarca() {
    return marca;
}
public void setMarca(String pMarca) {
    marca = pMarca;
}
```

```
private int edad;

public int getEdad() {
    return edad;
}
public void setEdad(int pEdad) {
    modelo = pEdad;
}
```

Propiedades (3/3)

En C# 3.0 y versiones posteriores, las propiedades implementadas automáticamente hacen que la declaración de atributos sea más concisa cuando no es necesaria ninguna lógica adicional en los descriptores de acceso. Cuando se declara una propiedad tal como se muestra en el ejemplo siguiente, el compilador crea un campo de respaldo privado y anónimo al que solo se puede acceder con los descriptores de acceso de propiedad **get** y **set**.

Propiedades en C#

```
public int id { get; set; }
public string nombre { get; set; }
public double ventas { get; set; }
```

Propiedades: Detrás del telón (3/3)

Las propiedades se pueden personalizar: validar datos, lanzar errores, etc.

```
class Person
{
    private string _name;  // the name field
    public string Name    // the Name property
    {
        get => _name;
        set => _name = value;
    }
}


class Employee
{
    private string _name;
    public string Name => _name != null ? _name : "NA";
}
```

```
public class Date
{
    private int _month = 7;  // Backing store

    public int Month
    {
        get => _month;
        set
        {
            if ((value > 0) && (value < 13))
            {
                _month = value;
            }
        }
    }
}
```

1. Elementos de una clase

Ejemplo

Diseñe e implemente una clase que represente un teléfono celular.

Pasos sugeridos en la clase anterior:

1. Hacer una lista con lo que se desea hacer.
2. Imaginar ejemplo y encontrar nombres para métodos.
3. Crear "esqueleto" y documentarlo.
4. Determinar los atributos necesarios.
5. Determ. los constructores y los parámetros necesarios.
6. Implementar los métodos.
7. Probar la clase.

Código fuente

2. Consejos en el diseño de clases

Tip: Always initialize data

C# won't initialize local variables for you, but it will initialize instance fields of objects. Don't rely on the defaults, but initialize the variables explicitly, either by supplying a default or by setting defaults in all constructors.

Tip: Not all fields need individual field accessors and mutators

You may need to get and set an employee's salary. You certainly won't need to change the hiring date once the object is constructed. And, quite often, objects have instance fields that you don't want others to get or set, for example, an array of state abbreviations in an Address class.

2. Consejos en el diseño de clases

Tip : Don't use too many basic types in a class

The idea is to replace multiple related uses of basic types with other classes. This keeps your classes easier to understand and to change. For example, replace the following instance fields in a **Customer class**...

```
private String street;
private String city;
private String state;
private int zip;
```

...with a new class called Address. This way, you can easily cope with changes to addresses, such as the need to deal with international addresses.

2. Consejos en el diseño de clases

Tip: Use a standard form for class definitions

We always list the contents of classes in the following order:

- public features
- package scope features
- private features

Within each section, we list:

- instance methods
- static methods
- instance fields
- static fields

After all, the users of your class are more interested in the public interface than in the details of the private implementation. And they are more interested in methods than in data.

However, there is no universal agreement on what is the best style. The Sun coding style guide for the Java programming language recommends listing fields first and then methods.

Whatever style you use, the most important thing is to be consistent.

2. Consejos en el diseño de clases

Tip: Break up classes that have too many responsibilities

This hint is, of course, vague: "too many" is obviously in the eye of the beholder. However, if there is an obvious way to make one complicated class into two classes that are conceptually simpler, seize the opportunity.

On the other hand, don't go overboard; 10 classes, each with only one method, is usually overkill. Here is an example of a bad design:

```
public class CardDeck { //bad design
    public CardDeck() { . . . }
    public void shuffle() { . . . }
    public int getTopValue() { . . . }
    public int getTopSuit() { . . . }
    public void draw() { . . . }

    private int[ ] value;
    private int[ ] suit;
}
```

2. Consejos en el diseño de clases

This class really implements two separate concepts: a deck of cards, with its shuffle and draw methods, and a card, with the methods to inspect the value and suit of a card. It makes sense to introduce a Card class that represents an individual card. Now you have two classes, each with its own responsibilities:

```
public class CardDeck {
    public CardDeck() { . . . }
    public void shuffle() { . . . }
    public Card getTop() { . . . }
    public void draw() { . . . }
    private Card[ ] cards;
}
```

```
public class Card {
    public Card(int aValue, int aSuit) { . . . }
    public int getValue() { . . . }
    public int getSuit() { . . . }
    private int value;
    private int suit;
}
```

Ejemplo:

Implementar las clases Card y CardDeck.

2. Consejos en el diseño de clases

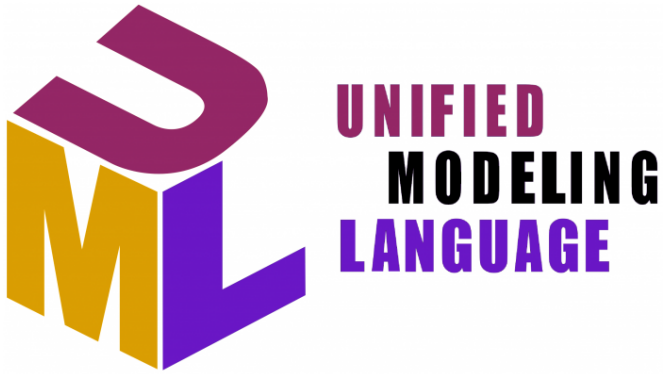
Tip: Make the names of your classes and methods reflect their responsibilities

Just as variables should have meaningful names that reflect what they represent, so should classes. (The standard library certainly contains some dubious examples, such as the Date class that describes time.)

A good convention is that a class name should be a noun (Order) or a noun preceded by an adjective (RushOrder) or a gerund (an "-ing" word, like BillingAddress). As for methods, follow the standard convention that accessor methods begin with a lowercase get (getSalary), and that mutator methods use a lowercase set (setSalary).

3. UML: Lenguaje unificado de modelado

El Lenguaje Unificado de Modelado (UML, por sus siglas en inglés) se estableció como un modelo estandarizado para describir aplicaciones que utilizan programación orientada a objetos (POO).



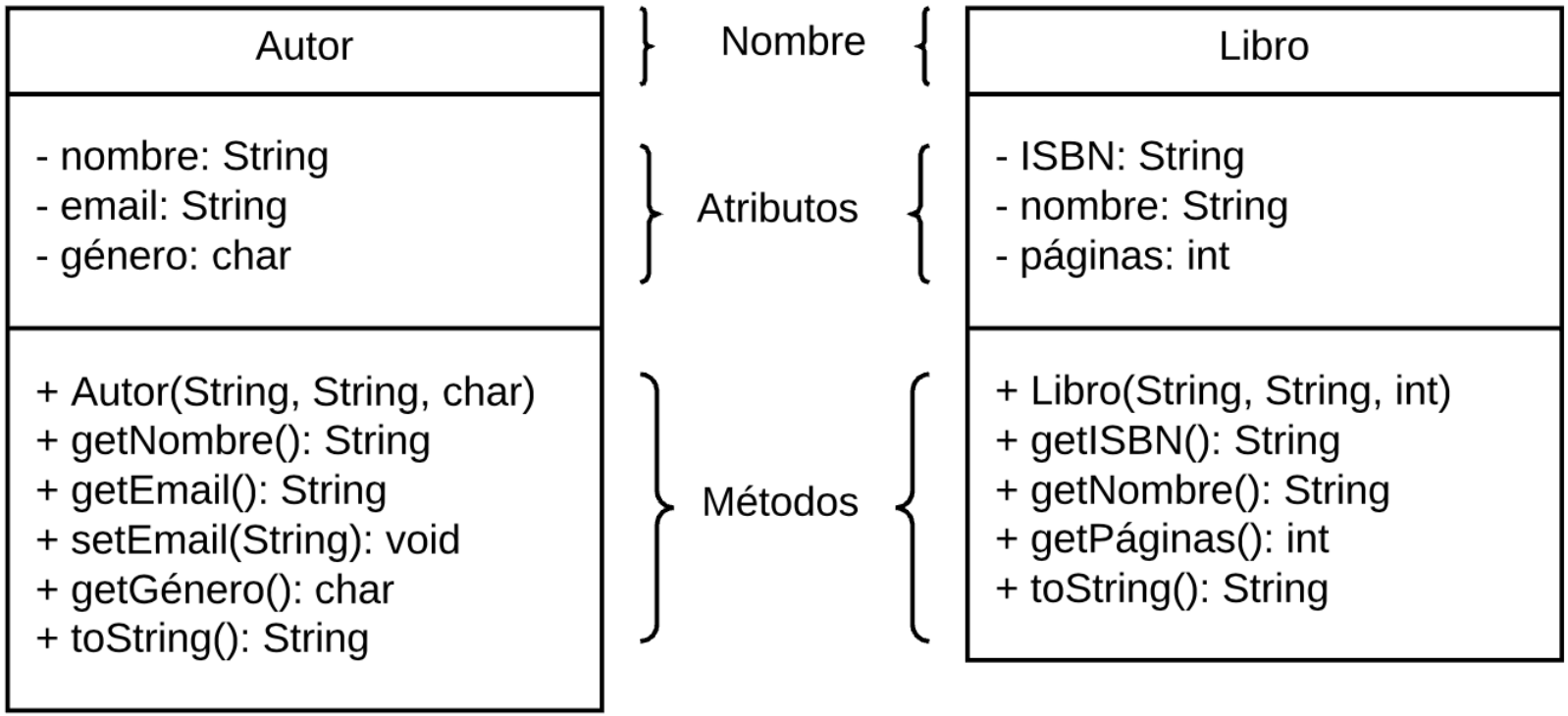
Ventajas:

- Es muy sencillo. Pese a que si es usado de forma completa puede llegar a complicarse, lo normal es que se simplifique.
- Es capaz de modelar todo tipo de sistemas.
- Es un lenguaje universal, haciendo que todos los miembros del equipo se relacionen a través de sus diagramas sean del ámbito que sean.
- Es visual y, por lo tanto, intuitivo.
- Es independiente del desarrollo, del lenguaje y de la plataforma.

3. UML: Lenguaje unificado de modelado

Diagrama de clases

El más popular en el UML. La figura de clase en sí misma consiste en un rectángulo de tres filas. La fila superior contiene el **nombre** de la clase, la fila del centro contiene los **atributos** de la clase y la última expresa los **métodos** o las operaciones que la clase puede utilizar.



3. UML: Lenguaje unificado de modelado

Componentes de un diagrama de clases

- **Nombre:** La primera fila en una figura de clase.
- **Atributos:** La segunda fila en una figura de clase. Cada atributo de una clase está ubicado en una línea separada.
- **Métodos:** La tercera fila en una figura de clase. Los métodos se organizan en un formato de lista donde cada uno posee su propia línea.

Modificadores de acceso

- Público (+)
- Privado (-)
- Protegido (#)
- Paquete (~)
- Estático (subrayado)

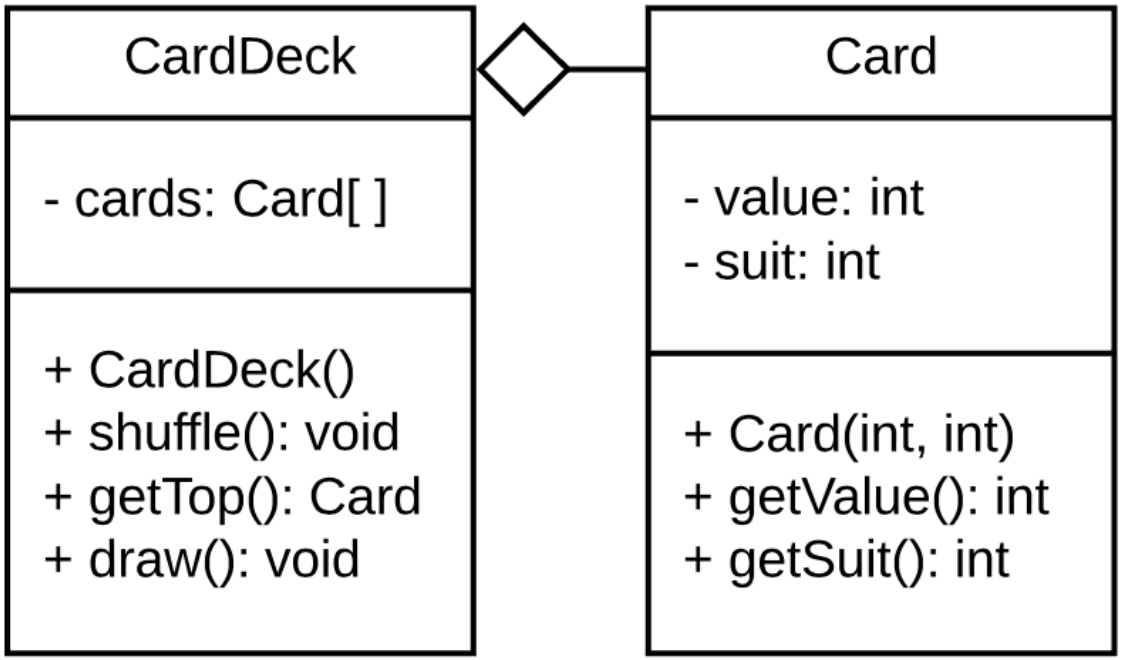
Ver desde el inicio hasta el 4:10 (Español)

Ver desde el inicio hasta el 3:35 (Inglés)

3. UML: Lenguaje unificado de modelado

Ejemplo:

Realizar el diagrama UML de clase para el ejemplo anterior.



4. Anuncios

Encargada de administración de clanes

- Instructora: María Morán.
- Correo electrónico: 00113417@uca.edu.sv

Favor verificar el Discord

Ya se encuentran las instrucciones para la práctica de laboratorio y las actividades optativas de la presente semana.

Unidad II: Programación orientada a objetos básica

Lic. Ronaldo Armando Canizales Turcios

Departamento de Electrónica e Informática, UCA

Ciclo virtual 01-2021